Enabling scientific teamwork

Mark Hereld[*], Randy Hudson[+], John Norris[+], Michael E Papka[*], Thomas Uram[*]
[*]Mathematics and Computer Science, Argonne National Laboratory, Argonne, IL 60439, USA
[+]Flash Center, University of Chicago, Chicago, IL 60622, USA

E-mail: papka@anl.gov

Abstract. The Computer Supported Collaborative Work research community has identified that the technology used to support distributed teams of researchers, such as email, instant messaging, and conferencing environments, are not enough. Building from a list of areas where it is believed technology can help support distributed teams, we have divided our efforts into support of asynchronous and synchronous activities. This paper will describe two of our recent efforts to improve the productivity of distributed science teams. One effort focused on supporting the management and tracking of milestones and results, with the hope of helping manage information overload. The second effort focused on providing an environment that supports real-time analysis of data. Both of these efforts are seen as add-ons to the existing collaborative infrastructure, developed to enhance the experience of teams working at a distance by removing barriers to effective communication.

1. Introduction

"Science is a collaborative enterprise, and the traditional environment for scientific collaboration is the laboratory. [1]" Scientists work together as part of a team to solve a common problem. Historically, researchers have worked in the same physical space, and, in the case of remote collaborations, traveled to the remote laboratory as needed in order to support the collaboration. Modern information technology has been diminishing the importance of a shared physical space. In the early 1990s William Wulf coined the term *collaboratory* in the National Research Council study "National Collaboratories: Applying Information Technologies for Scientific Research", where he laid out the vision of a research environment that was not bound by physical space, an environment that brings together researchers, resources and data to increase the scientific output and productivity of the teams [2]. Fifteen years later, do we have infrastructure to enable scientific teamwork?

Stepping back and looking at what being part of a team brings to scientific research yields a variety of different answers, from variance in expertise to a sounding board for the exchange of ideas. Dating back to the 1970s, research has shown distributed teams perform below their full potential [3]. The work of Cummings and Kiesler looked at the outcome of the National Science Foundation's Knowledge and Distributed Intelligence research program and found that teams that were distributed across multiple universities (locations) performed worse than their collocated counterparts in the generation of new ideas and knowledge [4]. They determined that technology such as email, instant messaging, or conferencing technology (both video and phone) did not provide any advantage to the distributed teams. Based on the data collected in the process of their research, they were able to identify some requirements of technology that might help change this outcome. These requirements included: (1) tools to manage and track the trajectory of tasks over time, (2) tools to reduce

information overload, (3) infrastructure to facilitate ongoing conversations, (4) infrastructure to encourage awareness with reasonable interruption for spontaneous talk, (5) tools that support simultaneous group decision making, and (6) tools and infrastructure capable of supporting presentations and meetings across a distance. Cummings and Kiesler went on to claim that "research on innovation in virtual organizations, and technology to support innovative teams, is critical to the future success of this type of work [5]."

The technology requirements laid out by Cummings and Kiesler can be viewed from a work mode of either asynchronous and/or synchronous tasks. Beginning with the asynchronous mode, the requirements of 1, 2, 3, and 6 represent tasks that can be done without direct interaction with other team members. Synchronous activities 2, 4, 5, and 6 require or could require direct interaction with other team members. This collection of efforts will address some of the requirements put forth in the previous paragraph and are examples of the use of technology to enable scientific teamwork. The rest of this paper will describe two of our recent efforts to improve the productivity of distributed science teams by addressing the technology requirements identified above. These efforts are described separately by whether they address asynchronous or synchronous teamwork.

2. Asynchronous tasks

We have been working with members of the University of Chicago's ASC / Alliance Center for Astrophysical Thermonuclear Flashes on the construction of tools to manage and track simulations and analysis over time, while reducing the amount of information the individual researchers must keep track of in an attempt to support an ongoing dialogue about the science being studied. The results of this effort are presented in this section.

We have created a simulation-analysis management system that, among other things, allows a user to set up, start, stop and monitor simulations; analyze simulation output; catalog and retrieve earlier simulations; and prepare output for publication. This work has been guided by some fundamental beliefs. We believe the system needs to account for maximum automation, both in terms of its interactions with scientists and its ability to easily accommodate new tasks. It needs to be based on a single control interface that is accessible from as many platforms as possible without a reduction in capabilities. The system must be built using open standards whenever possible, to ensure that it can interface with other infrastructure and be extended as needed. At all stages of the development, the end users must be engaged to allow for proper input into the process and ensure that all needs are met.

This section details the simulation-analysis management system. The system consists of components that capture and process simulation data and metadata *in situ* based primarily on log files generated during FLASH simulations, and a web application for interacting with the stored simulation data.

2.1. System components

The simulation-analysis management system consists of modular components to capture, store, verify, and process information during a simulation run. These components are assembled into workflows which we call *pipelines*. By examining the FLASH workflow, we have constructed a set of pipelines that capture the core FLASH processes. We also took care to preserve the modularity of the system, so that it could be easily employed for simulation-analysis management outside of FLASH.

Pipelines are used to collect data on, and create snapshots of, each recorded state of a simulation. They are also used to archive datasets created by a simulation and extract data for post-processing. Pipelines consist of the simulation code, a collector component, and an archiver component. The first process of the pipeline is the simulation itself. The *collector* and *archiver* processes follow next in the pipeline.

The collector is run on the system hosting the simulation. This program tracks the progress of the simulation via a central log file, and notes the current timestep that the simulation is in. All other data is collected by dynamically loaded tools, which the collector calls for each line of the central log file, as well as separate calls to mark the beginning and end of timesteps. The collector is started anytime

after the simulation has been queued to run, even long after the simulation has ended. The program is capable of recovering from fatal conditions (e.g. being killed by an administrator). It can also verify the data integrity of a previously collected simulation. The user receives email notification of significant events, including simulation completion, simulation crash, and collector death.

The archiver is run on the system hosting the files to be archived, preferably by the creator of the files. This program tracks the progress of the simulation via the database, archiving files as they are added to the database. It can be started any time after the collector has been started, and may, in the future, be started by the collector. Like the collector, the archiver is capable of recovering from fatal conditions. The user receives email notification of significant events such as archival completion and archiver death.

We introduce "soft" constraints, or suggestions, wherever appropriate to urge or remind users to use standard forms and categories wherever they can. For example, the collector prompts the user for a run description at run time to encourage the entering of a meaningful description versus a cut and paste of previous entries. The collector and archiver can use some sensible defaults for data they otherwise need from the user; for example, if given no destination path, the archiver will construct a unique path in a standard way.

In addition to the collector and the archiver, two additional supporting tools have been developed based on the needs of the users: the *associator* and the *verifier*. The associator is a simple command-line utility that allows the user to associate files, usually post-processing results, to a previously collected simulation and record the association in the database. Any files connected to a simulation in this manner also become available for archiving. The verifier represents the first of a generation of programs meant to maintain the accuracy of the information in the database. When run on a system which hosts files referenced in the database, the verifier confirms that each local file actually exists and updates the database as needed.

The database was designed to be as independent as possible from the peculiarities of a given simulation or domain science so that the system could be reused. The database stores meta-data from several simulation files, user preferences, names and locations of data files (e.g. output from the VisIt visualization tool), and a catalog of old simulations and links to files that users consider important (e.g. scripts and movies). Django, a Python-based web framework, runs on the server, queries the database and returns results to the web browser. Django uses Matplotlib to create data plots and return them to the browser. Currently, users annotate simulations and runs with Django's administrative component.

To minimize the effort to make the FLASH pipeline usable by all computational scientists, the FLASH-particular code of the collector is isolated from its control module in the set of dynamically loaded tools. File types are defined in a database table rather than as a database enumerated type. Variables and their names are in separate tables of the database.

2.2. End user application

The end-user application is a web application that acts as a single place from which the computational scientist can manage most preparations for a simulation, the simulation itself, and post-processing and analysis. The core functionality of the web application (Figure 1) includes the following:

- Set up, start, stop and monitor simulations
- Keep track of old simulations
- Edit annotations of simulations
- Launch scripts and view their output
- Prepare plots for publication
- Send plot data to other plotting tools
- Store simulation parameters and file locations
- View meta-data from the database
- Set user preferences

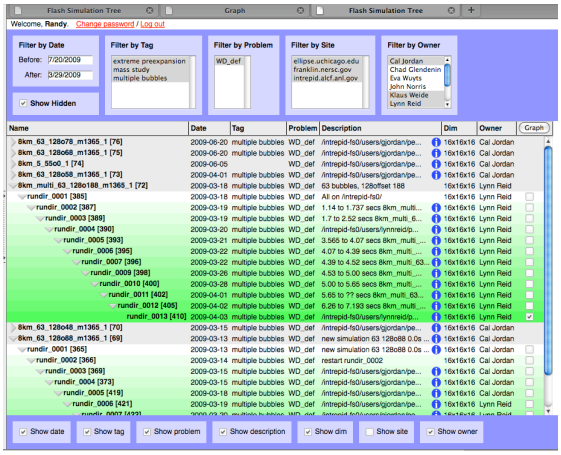

Figure 1. Screen capture of web application that shows a given simulation and its multiple restarts. Users can filter which results are visible by choosing from a variety of different options at the top of the page.

This section describes the current state of the web application.

Simulations are composed of one or more individual runs, exactly one of which is the initial run. All other runs in the simulation are restarts from an earlier run. Since this naturally lends itself to a hierarchical representation, the application uses a tree structure to present the simulations in the database to the user. Each restart is displayed as a child of its parent run (the run that generated the data used for the restart). The user can filter out simulation trees by date, owner, problem type, and several other criteria; navigate through the individual runs of each simulation; and select which runs to generate graphs from. The user can also choose which columns of the table are displayed and, if logged in, can edit tags, names and descriptions of runs and simulations.

When the user selects one or more runs and clicks the graph button, a curve for each simulation path containing a selected run is plotted on a graph page. Multiple data series can be displayed on the graph, with optional operations performed on the data prior to plotting, such as addition, subtraction, multiplication, division, modulo and two types of accumulation. The division operation, for example, can be used to convert mass units from grams to solar mass. An accumulator can integrate a number of blocks over time steps. The display of graphs can be customized according to several criteria, such as data series selection, plot labels, the number of variables to display on the vertical axis, and whether run delimiters should appear. As mentioned previously, simulations often consist of multiple compute runs, with the results aggregated from all runs; run delimiters indicate these run boundaries on the graph, and can help the researchers identify problems that may have arisen between runs.

Graph data and display characteristics are encoded in the graph URL, so that the URL can be embedded in external sources (e.g. web pages) where they will be regenerated when accessed. Typically, graph URLs are constructed to capture a static view of the data, in which case the plots do not change over time. When monitoring ongoing simulations, however, FLASH researchers can view graphs generated from the most recent simulation data. These graphs can be integrated with project pages to provide current simulation state in the full context of a project.

This development has been guided by regular interactions with FLASH scientists using the system. As such, it is tailored to address many aspects of their everyday work, and reveals what they perceive as requirements for performing their science in a collaborative manner. Working in the system, users are aware of the activities of their cohorts on shared and non-shared projects. Outside the system, in their wiki-based project portal, they can view snapshots of data from past simulations and dynamically updated graphs of running simulations.

3. Synchronous tasks

In terms of synchronous tasks we have been working to minimize the information overload that users often experience in terms of managing a number of applications as part of a single task: providing infrastructure to increase the sense of presence of remote collaborators, interactive visualization tools that support real-time group investigation of even the largest datasets, and infrastructure to support traditional activities that occur when a group is separated by distance.

Many of these tasks build on the Access Grid infrastructure developed over the past ten years, which in turn arise from a long history of collaboration infrastructure [6,7]. By leveraging the Access

Grid, users have a single integrated point to connect to remote colleagues, organize data, and launch applications to interact with the data. Via the Access Grid, users can see if remote colleagues are active, much like standard instant messaging technology in use today. In addition, since the initial deployment of the Access Grid, it has had support for remote presentations and meetings. With over 50,000 downloads and deployments in more than seventy five different countries, the Access Grid provides a platform to enable scientific teamwork.

In order to provide infrastructure for distributed collaborations to make real-time decisions, we have developed VPCScreenIX. VPCScreenIX is an application for distributing screen content to multiple simultaneous remote users, allowing them to interact with the originating application as if it were running locally. While there have previously been many similar efforts, VPCScreenIX excels in its network efficiency, due to its use of streaming video as a delivery mechanism, and its ability to scale to many simultaneous users. These concerns are vital in the context of HPC-based visualization, to maximize the involvement of researchers who are often distant from the originating visualization applications and the huge data and compute resources they require.

3.1. Related efforts

Much effort has been dedicated to delivering remote access to computers and applications. Prominent among these efforts is VNC [8], for its performance, stability, openness, and wide availability. Remote Desktop is another notable effort, with an open specification and clients on many platforms, but server support typically limited to Windows. Many proprietary solutions exist, but have been omitted here.

VNC provides remote desktop access using a server and one or more clients. VNC uses a variety of encodings to optimize for bandwidth or quality. VNC uses the Remote Frame Buffer (RFB) protocol, an open specification that has been implemented in many applications on all major operating systems.

VPCScreen captures content from the screen and streams it as video to remote viewers. The video is encoded using a version of the H.261 codec modified to support arbitrary size video frames, called H.261as. The H.261 standard supports CIF (352x288) and QCIF (176x144) resolutions. H.261as overcomes this resolution limitation by applying the standard H.261 encoding to larger frames.

The ParaView Streaming Plug-in, developed at Argonne National Laboratory, enables researchers to share and discuss visualizations of their data by extending ParaView – an open source visualization application that runs on desktops as well as supercomputers – to stream its render area as video to remote receivers. The plug-in uses the H.261as codec described above. The user loads the plugin, specifies the stream destination address, begins streaming, and proceeds with normal use of the ParaView application. Remote sites start the viewing application – vic – with the same destination address to view the ParaView video stream.

3.2. Application sharing

VPCScreenIX is a remote desktop application that uses streaming video to deliver content to remote viewers. Our goal was to develop an application-agnostic version of the ParaView Streaming Plug-in so that any application could be shared without modification. Streaming video was chosen as a delivery mechanism primarily because video codecs have been rigorously optimized to maximize framerate while minimizing bandwidth. Remote interaction support was adapted from the synergy2 project, which has largely solved the problem of cross-platform mouse/keyboard capture and synthesis.

The complete system (Figure 2) consists of three components: the IX library for capturing and synthesizing mouse and keyboard events; a video tool, vic, modified to support capture of mouse and keyboard events; and a modified version of VPCScreen that listens for events from remote viewers running vic.

Building on the mouse/keyboard sharing tool synergy2, we developed an interaction library that solves a longstanding problem in collaborative applications: capturing mouse and keyboard events and sending them to remote systems, where they are synthesized into the system as if they occurred locally, while managing the complexity of this task across multiple platforms. This library includes an

IXSender component, which can track windows on the display and capture mouse and keyboard events that occur inside them. The IXSender transmits these changes over TCP to the remote application, which has an integrated IXReceiver. The IXReceiver synthesizes these events into the local system. In the case of VPCScreenIX, events from the IXReceiver are input into the application whose window is being captured and transmitted. These events are processed in the context of the target application, effecting the intended actions.

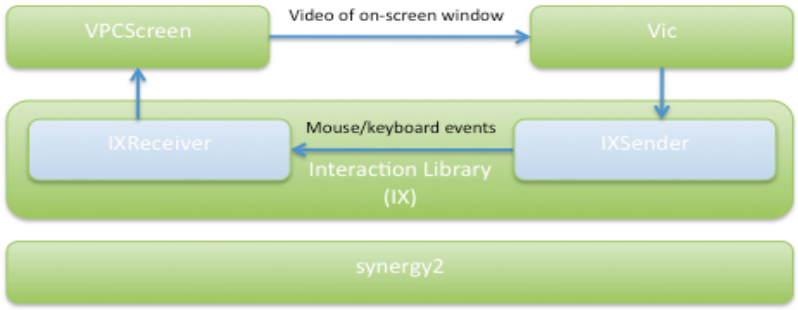

Figure 2. Architecture of the IX interaction library and its integration into VPCScreen and vic.

Users can select any visualization application (e.g., VisIT, VisTrails) from their desktop and stream its window as video. Furthermore, any application can be streamed, not just visualization applications, and the applications require no modification. The screenshots in Figure 3 show ParaView on MacOS being displayed on a Windows desktop.

We have integrated VPCScreenIX with the Access Grid to simplify starting applications. This integration enhances the existing remote collaboration by allowing participants to view and discuss the visualization, and does so with little user interaction. The application also benefits from the Access

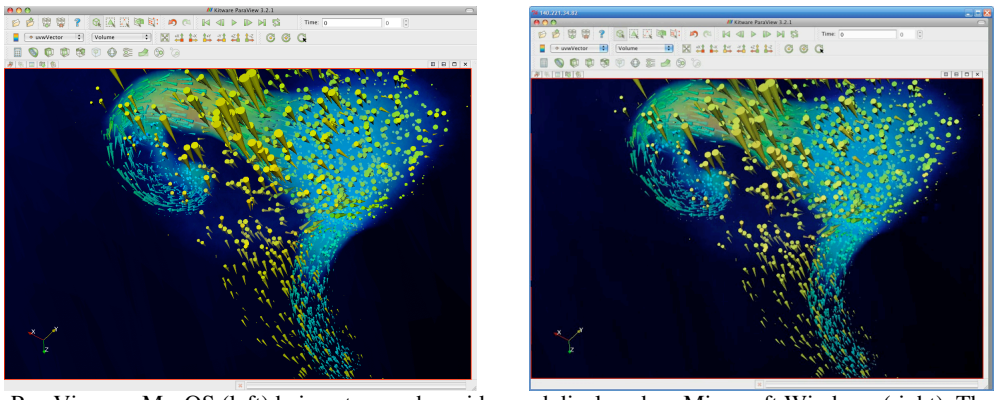

Figure 3. ParaView on MacOS (left) being streamed as video and displayed on Microsoft Windows (right). The Windows user can interact with the remote application natively.

Grid's intelligent bridging infrastructure, a transparent substitute for multicast networking.

VPCScreenIX is integrated as a plugin to the Access Grid Venue Client. The plugin adds buttons to the Venue Client toolbar for streaming a window, a region, or the entire screen. Clicking on one of these buttons launches VPCScreenIX in the selected mode, prompting the user to select a window or screen region as needed, and then begins to stream screen content to the established Access Grid video destination address.

The modified version of vic is included in the VPCScreenIXConsumerService, which replaces the standard video service for the Access Grid. It can view AG-native video streams and also supports interaction with remote sources when available. Using this service, users can see the video of their remote collaborators and video transmitted by VPCScreenIX, and interact with the VPCScreenIX streams with a single click.

3.3. Performance analysis

We studied the efficiency of VPCScreenIX versus VNC by devising a test to transmit a fixed size buffer with random noise variation in the output. The use of a random noise pattern is a fair approximation of the activity during use of a visualization application, with a significant portion of the screen content changing fairly rapidly. The machines used to conduct these tests were configured as follows:

- Sending machine: OSXvnc, MacOSX 10.5.6, 2.33 GHz Intel Core 2 Duo, 3GB RAM, ATI Radeon X1600
- Receiving machine: RealVNC, Windows XP, 2 x 3.4GHz Pentium, 2GB RAM, nVidia GeForce 7800 GTX
- VNC configuration: ZRLE (Zlib compressed Run Length Encoding), 32 bits per pixel

Measurements were made on the Mac using Apple's Instruments software.

In our tests, we found that VPCScreenIX required about one fifth of the bandwidth required by OSXvnc. Processor utilization between the two was comparable, with VPCScreenIX requiring on average about 10% more CPU than OSXvnc.

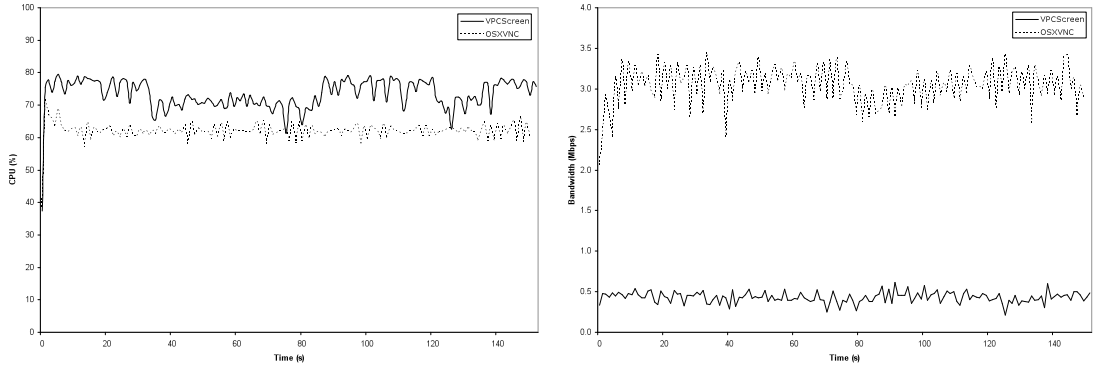

Figure 4. Comparison of CPU usage (left) and bandwidth usage (right) of VPCScreenIX and VNC. Measurements were taken using a transmitted image size of 1600x1200, while a 1450x820 video noise pattern was displayed.

3.4. User testing

Using the AccessGrid, we conducted a test of VPCScreenIX with climate scientists at Argonne National Laboratory (ANL) and Los Alamos National Laboratory (LANL), using ParaView to view an ocean modeling dataset. The scientists interacted with the dataset, using the Access Grid audio to discuss what was being shown and how to modify the visualization, while sharing a synchronized view of the application. The remote scientist was, in fact, unfamiliar with ParaView, but was successfully guided by the other through the steps necessary to develop the view of the data he desired.

This test was conducted using ParaView 3.4, running on Windows XP and transmitting over the esNET network. The receiving vic was run on Windows XP. The resolution of the video stream was 1619x864.

Following this test, the users concluded that the ability to share arbitrary applications with remote colleagues using VPCScreenIX represented a dramatic shift from their normal interactions, in which they typically use PowerPoint to view static images of their data visualizations. They said that it would be helpful for them to cooperatively develop their visualizations, and that it would contribute to a better shared understanding of their data and the problems at hand.

4. Conclusions

Science is a group effort and it requires technology that supports the construction of the best scientific teams regardless of the team members' locations. We believe that as scientific efforts become larger and more distributed, technology should be used to break down the barriers that hinder the productivity of teams working together at a distance. Guided by results from the computer supported

collaborative work research community, we have researched, developed and deployed tools and infrastructure that address the areas of greatest need. We have described our efforts in terms of enabling asynchronous and synchronous teamwork. We have presented tools to help keep groups in sync by providing a framework for annotating results and to have the results automatically refreshed. We have built tools that allow for the real-time sharing of results to enable a decision-making environment that is independent of the location of team members. We have leveraged the Access Grid environment to simplify the integration and deployment of much of the described technology. The Access Grid also provides a familiar environment in which applications, data and colleagues are integrated together. In our ongoing collaboration with science users, we have begun to overcome obstacles to effective science in distributed teams.

5. Acknowledgements

This work has been supported in part by the DOE, under contract DE-AC02-06CH11357, the NSF by grant OCI-0222509, NLM by contract N01-LM-3-3508 and the DOE-supported ASC / Alliance Center for Astrophysical Thermonuclear Flashes at the University of Chicago.

6. References

[1] T. A. Finholt, G. M. Olson, *From Laboratories to Collaboratories: A new Organizational Form for Scientific Collaboration*, **Psychological Science**, 8(1), pp. 28 – 26, 1997.

[2] V.G. Cerf, et al., **National Collaboratories: Applying Information Technologies for Scientific Research**, National Academy Press, 1993.

[3] I. D. Steiner, **Group Processes and Productivity**, Academic Press, 1972.

[4] J. N. Cummings and S. Kiesler, *Collaborative Research Across Disciplinary and Organizational Boundaries*, **Social Studies of Science**, 25(5), pp. 703-722, 2005.

[5] J. N. Cummings, S. Kiesler, *Who Collaborates Successfully? Prior Experience Reduces Collaboration Barriers in Distributed Interdisciplinary Research*, **Proceedings Of The ACM 2008 Conference On Computer Supported Cooperative Work**, pp. 437-446, 2008.

[6] T. L. Disz, R. Evard, M. W. Henderson, W. Nickless, R. Olson, M. E. Papka, and R. Stevens. *Designing The Future Of Collaborative Science: Argonne's Futures Laboratory*, **IEEE Parallel and Distributed Technology**, 3(2), pp.14–21, 1995.

[7] R. Stevens, *Access Grid: Enabling Group Oriented Collaboration On The Grid*, **The Grid: Blueprint for a New Computing Infrastructure**, pp. 191–199, Morgan Kaufmann, 1999.

[8] T. Richardson, Q. Stafford-Fraser, K.R. Wood, A. Hopper, *Virtual Network Computing*, **IEEE Internet Computing**, 2(1), pp. 33-38, 1998.